UNITED STATES PATENT APPLICATION

for

A METHOD AND AN APPARATUS FOR INTERLEAVING READ DATA RETURN IN A PACKETIZED INTERCONNECT TO MEMORY

Inventors:

Hemant G. Rotithor

An-Chow Lai

Randy B. Osborne

Olivier C. Maquelin

Mladenko Vukic

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard
Los Angeles, CA 90025-1030
(408) 720-8300

Attorney's Docket No.: 042390.P16971

"Express Mail" mailing label number: EV409359803US
Date of Deposit: January 29, 2004
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to Mailstop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Esther Campbell
(Typed or printed name of person mailing paper or fee)

Esther Campbell
(Signature of person mailing paper or fee)

January 29, 2004
(Date signed)

A METHOD AND AN APPARATUS FOR INTERLEAVING READ DATA RETURN IN A PACKETIZED INTERCONNECT TO MEMORY

FIELD OF INVENTION

[0001] The present invention relates to computer systems, and more particularly, to routing read data return in a computer.

BACKGROUND

[0002] In a typical computer system, memory page misses incur a high latency in returning data in response to read requests. Interleaved memory channels can process back to back memory page misses in parallel and overlap the latency from the two page misses over a longer burst length. In comparison, lock step memory channels process page misses sequentially over shorter burst length. . Interleaved memory channels thus have higher efficiency of handling access patterns with many page misses than lock step memory channels. In general, applications that have a significant number of page misses perform better with interleaved memory channels.

[0003] Typically, each interleaved channel independently processes a read request and returns read data using half the peak memory system bandwidth. A read request, also known as a read, commonly causes a cache line of data to be returned from the memory. Returning read data at half memory system bandwidth implies that the latency to return the last byte in the cache line is higher compared to the case in which the cache line is returned from two channels in lock step. When access patterns have many memory page hits, interleaved channel memory performance degrades if the read requests sent to the interleaved channels are not well balanced.

[0004] A software program may make a read request from a central processing unit (CPU) for different data sizes starting at the granularity of a byte. If the data requested is not in the CPU cache, the read request is sent to the memory to retrieve the data. Although, the original read may request data in a certain unit smaller than a cache line, such as, for example, a byte, a word, a double word, etc., the CPU retrieves a cache line of data from the memory in response to the read request because of locality of spatial references. The size of a cache line varies from system to system, e.g., 64 bytes, 128 bytes, etc. The cache line of data is handled in the CPU core at the granularity of a chunk, which is smaller than the cache line size, which may be 8 bytes, 16 bytes, etc. The data that the application program originally requested is contained in one of the chunks of the cache line called the *critical chunk*. A read request stalls in the CPU for the critical chunk, and therefore, reducing the latency of the critical chunk improves the performance of the system. To reduce the latency of the critical chunk, the memory system returns the critical chunk in a cache line first in the stream of bytes returned in response to a read request. Furthermore, reducing latency of the non-critical chunks of the cache line may improve performance for some applications because the CPU core may have other requests that ask for the other data bytes in the cache line.

[0005] Cache lines returned in response to the read requests are typically sent via an interconnect from a memory controller to the CPU. A packetized interconnect sends packets of messages containing information over a link layer and a physical layer. Packets emitted by the CPU contain requests to the memory and cache line data for write requests. Packets received by the CPU include read responses containing cache line data. At the link layer, a packet may be organized into equal sized flits for efficient

transmission. A flit is the granularity at which the link layer of the packetized interconnect sends data.

[0006] Currently, data from interleaved memory channels is sent via a shared front side bus (FSB) to the CPU, such as a P4FSB. On the shared FSB, read data return may be sent as soon as it becomes available from a memory channel and the transfer may be interrupted by inserting wait states until more chunks of data become available. This technique reduces the latency to the critical chunk of the cache line if not all the read data return is available, or is available at lower bandwidth than the FSB can deliver. Currently, the P4FSB protocol allows data received in response to only one read request to be returned at any given time, and thus, cache lines corresponding to two read requests simultaneously returning from two memory channels are sent sequentially.

[0007] On a packetized interconnect, a cache line of read data is stored and forwarded as illustrated in Figures 1A and 1B. In response to a read request, chunks of data of the read return are stored temporarily in a buffer. In this application the read returns are assumed to be stored in a FIFO buffer in order of return from the memory controller and top of the read return queue means the head of this FIFO, or oldest pending read return. Once enough chunks of data of a cache line have accumulated, a header and the chunks are sent in a stream to the CPU in a packet without interruption. The header is sent contiguously with the packet. Store and forwarding is necessary to send cache line data in one packet. Although chunks of a second cache line may be available from another memory channel, the chunks of the second cache line are not sent until all the chunks of the first cache line have been sent.

[0008] The above practice is a simple, but is a low performance, option because there is a store and forward delay in sending the critical chunk after it is received from the memory channel as the critical chunk sits in the read return buffer. Furthermore, simultaneously arriving read returns are serialized on the interconnect by buffering the read returns immediately following the first one. Thus, there is additional delay in sending these read returns. As a result, a larger overall latency is incurred.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention will be understood from the detailed description that follows and from the accompanying drawings, which however, should not be taken to limit the appended claims to the specific embodiments shown, but are for explanation and understanding only.

[0010] Figure 1A shows a flow diagram of a prior art process for forwarding data in response to a read request.

[0011] Figure 1B shows a timing diagram of an example of data transfer according to store-and-forward.

[0012] Figure 2 shows an exemplary embodiment of a computer system.

[0013] Figure 3A shows a flow diagram describing one embodiment of a process for forwarding data in response to read requests.

[0014] Figure 3B illustrates an example of data transfer according to one embodiment of critical chunk with bubble.

[0015] Figure 4A shows a flow diagram describing one embodiment of a process for forwarding data in response to read requests.

[0016] Figure 4B illustrates an example of data transfer according to one embodiment of critical chunk interleaving.

[0017] Figure 5A shows a flow diagram describing one embodiment of a process for forwarding data in response to read requests.

[0018] Figure 5B illustrates an example of data transfer according to one embodiment of flit-level interleaving.

[0019] Figure 5C illustrates another example of data transfer according to one embodiment of flit-level interleaving.

[0020] Figure 6A shows the logical representation of an embodiment of a memory controller hub performing flit-level interleaving.

[0021] Figure 6B illustrates one example of data transfer according to one embodiment of flit-level interleaving.

[0022] Figure 6C illustrates another example of data transfer according to one embodiment of flit-level interleaving.

[0023] Figure 6D illustrates another example of data transfer according to one embodiment of flit-level interleaving.

DETAILED DESCRIPTION

[0024] A method and an apparatus to process read data return is described. In one embodiment, chunks of a first cache line and a second cache line are interleaved. Each cache line has a critical chunk. The critical chunks of the first and second cache lines appear in an interleaved stream before the non-critical chunks of the first and second cache lines. The interleaved chunks of the first and second cache lines are sent via a packetized interconnect to a processor. Some examples of data transfer according to various embodiments of the present invention are shown in Figures 3B, 4B, 5B, 5C, 6B, 6C, and 6D, and details of which are described below.

[0025] In the following description, numerous specific details are set forth. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures, and techniques have not been shown in detail in order not to obscure the understanding of this description. Furthermore, references to "one embodiment" in the current description may or may not be directed to the same embodiment.

[0026] Figure 2 shows an exemplary embodiment of a computer system 200. One should appreciate that different embodiments of the system may include additional components not shown in Figure 2. System 200 includes a CPU 210, a memory controller hub (MCH) 220, and two dynamic random access memory (DRAM) channels 230 and 240. In one embodiment, the DRAM channels 230 and 240 are coupled to a number of DRAM devices (not shown). One should appreciate that other types of memory and memory channels may be used in various embodiments, such as, for example, synchronous DRAM (SDRAM), double data rate (DDR) SDRAM, etc.

[0027] The CPU 210 and the DRAM channels 230 and 240 are coupled to the MCH 220. In one embodiment, the CPU 210 is coupled to the MCH 220 by an outbound packetized link 212 and an inbound packetized link 214. In response to a read request in a program being executed by the CPU 210, the CPU 210 sends a read request via the outbound packetized link 212 to the MCH 220. In response to the request, the MCH 220 retrieves data from one of the DRAM channels 230 and 240. In one embodiment, the data is returned as a cache line. The MCH 220 returns the data to the CPU 210 via the inbound packetized link 214 as described in more details below.

[0028] In one embodiment, the cache line has a size of 64 bytes. The cache line may be split into a number of chunks. For example, in one embodiment, a cache line of 64 bytes is split into 8 chunks, each chunk having 8 bytes. However, one should appreciate that the chunk size varies in different systems. The cache line returned may include data in addition to what is actually requested by the program because the data requested by the program may be less than a cache line, such as, for example, a byte, or a word. The chunk containing the data actually requested is referred to as a critical chunk.

[0029] In one embodiment, the data is sent in packets on the inbound packetized link 214 in units at the granularity of a flit. A flit is the granularity at which link layer of the packetized interconnect sends data. The flit is a non-interruptible unit of data sent on a communication medium between the CPU 210 and the interconnect 214. The size of the flit varies among different embodiments, for example, a flit size may be 8 or 4 bytes. A chunk may be sent in one or more flits. One should appreciate that the flit size may or may not be the same as the chunk size. Furthermore, the time to send a flit depends on

the link speed and link width. In one embodiment, a read or write request packet is sent in one flit, while a read or write cache line data packet is sent in multiple flits.

[0030] Referring to Figure 2, the MCH 220 includes a link buffer 222, a read buffer 224, a write buffer 226, an arbiter 228 that arbitrates between reads and writes, two channel controllers 250 and 260, read data return circuitry 270, and a packetized interconnect interface 280. In one embodiment, the circuitry 270 includes two read return buffers 272 and 274 and a multiplexer 276. A request from the CPU 210 is forwarded to the MCH 220 via the outbound packetized link 212 and is temporarily held in the link buffer 222. The request may be a read request or a write request. The read request is forwarded to the read buffer 224 to be input to the arbiter 228. Likewise, the write request is forwarded to the write buffer 226 to be input to the arbiter 228. The arbiter 228 forwards either the read request or the write request to one of the channel controllers 250 and 260, based on some mapping functions.

[0031] The channel controllers 250 and 260 are coupled to the DRAM channels 230 and 240 respectively. In one embodiment, each DRAM channel has a dedicated channel controller. In an alternate embodiment, a channel controller handles multiple DRAM channels. A read request for data from the DRAM channel 230 is forwarded from the arbiter 228 via the channel controller 250 to the DRAM channel 230. In response to the read request, the DRAM channel 230 returns a cache line of data to the MCH 220 via the circuitry 270. Likewise, a read request for data from the DRAM channel 240 is forwarded via the channel controller 260 to the DRAM channel 240. In response to the read request, the DRAM channel 240 returns a cache line of data to the circuitry 270.

[0032] Referring to Figure 2, the circuitry 270 includes two read return buffers 272 and 274 and a multiplexer 276. The chunks of data returned from the DRAM channels 230 and 240 are forwarded to the read return buffers 274 and 272 respectively. Alternatively, instead of two buffers 274 and 272, a single buffer may be used to buffer both data returned from the DRAM channel 230 and the DRAM channel 240. Referring to Figure 2, the read return buffers 272 and 274 are coupled to the inputs of the multiplexer 276. In one embodiment, the multiplexer 276 selects data a flit at a time from either of the read return buffers 272 and 274 and outputs the selected data. The packetized interconnect interface 280 outputs the selected chunks to the CPU 210 via the inbound packetized link 214.

[0033] In one embodiment, the channel controllers 250 and 260 are substantially identical. Referring to Figure 2, the channel controller 250 includes a scheduler 251, a read buffer 253, and a write buffer 255 which may be shared between the channels. Similarly, the channel controller 260 includes a scheduler 261, a read buffer 263, and a write buffer 265. The read buffers 253 and 263 store read requests temporarily and input the read requests to the schedulers 251 and 261 respectively. Likewise, the write buffers 255 and 265 store write requests temporarily and input the write requests to the schedulers 251 and 261 respectively. The schedulers 251 and 261 schedule transmission of read requests and write requests to the DRAM channel 230 and the DRAM channel 240 respectively.

[0034] In one embodiment, the packetized interconnect 214 runs faster than the DRAM channels 230 and 240. For example, the interconnect 214 may run on an interconnect packet clock frequency that delivers a bandwidth of 10.6 GB/s in each

direction while each of the DRAM channels 230 and 240 runs at a clock frequency that delivers a bandwidth of 5.3 GB/s. Therefore, the packetized interconnect 214 may send data faster than receiving data from either of the DRAM channels 230 and 240. As a result, there may be a mismatch between the rate at which chunks are produced and the rate at which the chunks are consumed. Such a mismatch is not desirable if the data is to be sent in a contiguous packet. However, embodiments of the present invention take advantage of this mismatch to send data efficiently. Three exemplary embodiments are described in details below.

Critical Chunk with Bubble

[0035] One exemplary embodiment of a process for forwarding read return data is referred to as critical chunk with bubble, which includes sending a critical chunk when the critical chunk becomes available, storing the non-critical chunks, and sending the non-critical chunks in another packet. Figure 3A shows a flow diagram of one exemplary embodiment of critical chunk with bubble and Figure 3B illustrates an example of data transfer according to the critical chunk with bubble. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. Processing logic buffers a chunk of data from a storage device, such as, for example, one of the DRAM channels 230 and 240 in Figure 2 (processing block 305). Then processing logic checks whether the read return on the top of a read return queue has any critical chunk not yet forwarded to the CPU 210 (processing block 310). If the cache line of the read on top of the read return queue has a critical chunk not yet forwarded, then processing logic checks whether a header has been sent (processing

block 312). If the header has been sent, processing logic gets the critical chunk for the read return on the top of the read return queue and sends the critical chunk on the interconnect (processing block 314). Otherwise, processing logic sends the header and sets the flag "header sent" to 1 (processing block 316). Processing logic then repeats processing block 305. One should appreciate that the oldest read, which is a request for data coming into MCH 220, may not correspond to the read return at the top of the read return queue from the MCH 220. In other words, the read requests and read returns may be in different orders.

[0036] If the critical chunk of the cache line of the oldest read return has been forwarded, then processing logic checks whether enough chunks of the read return on the top of the read return queue have accumulated (processing block 320). If there are enough chunks accumulated, then processing logic starts sending chunks of the cache line of the read return on the top of the read return queue onto the interconnect (processing block 323). In one embodiment, processing logic waits until all non-critical chunks of the read at the top of the read return queue have accumulated to send the chunks via the interconnect in a single transfer without interruption. Processing logic checks whether all the chunks of the cache line of the read at the top of the return queue have been sent (processing block 325). If not, then processing logic repeats processing block 305. Otherwise, processing logic removes the read return on the top of the read return queue from the queue (processing block 327). Processing logic then repeats processing block 305.

[0037] Referring to Figure 3B, two exemplary cache lines 610 and 620 corresponding to two read returns that arrive in an overlapping manner via two memory

channels from two storage devices, such as, for example, the DRAM channels 230 and 240 in Figure 2. The example 650 illustrates a stream of chunks in the critical chunk with bubble scheme. The memory clock 600 is shown above the read returns 610 and 620. For the purpose of illustration, the following discussion assumes that the memory clock 600 in Figure 3B is at 333MHz (for a two-channel DDR2 667) and the frequency of the flit clock is 1333 MHz. Suppose the cache line 610 is the data for the read at the top of the read return queue of read returns in the current example. The critical chunks 652 of the cache line 610 are forwarded when the critical chunks 652 become available. The rest of the cache line 610 is stored and not forwarded to the interconnect 214 (referring to Figure 2) until 654, at which time the remaining cache line can be streamed to the interconnect 214 in one packet without interruption. Referring to Figure 3B, the earliest time to deliver the third chunk of the exemplary cache line 610 is substantially equal to the time at 608 minus 6 interconnect cycles so that there is no bubble when the rest of the cache line 610 is transferred on the interconnect. The data 656 including the second cache line 620 and a header 658 is forwarded after the transmission of the data 654 of the cache line 610 has been completed. In one embodiment, the time gap between sending the flits 652 and the flits 654 is used to send the flits of a prefetched cache line of another read data return in order to increase the overall efficiency and performance of the system. The prefetched cache line may be a result of the read in 214 (referring to Fig. 2) hitting an address in the write buffer 226 and getting its data forwarded or because of a read hitting an address in a prefetch data buffer when the MCH 220 has a chipset prefetcher (not shown).

[0038] In one embodiment, two types of packets are defined for transferring the chunks, namely, a critical chunk packet and a cache line packet. By sending a critical chunk when the critical chunk becomes available and storing the rest of the cache line to be forwarded later, the latency to the critical chunk is reduced. For example, referring to Figure 3B, the critical chunk 652 of the read 610 is sent approximately one and half memory clock cycle earlier than the corresponding critical chunk 662 sent using the store and forward scheme 660. However, the cache line latency and the latency to the other reads in the case of simultaneously arriving reads is still high.

Critical Chunk Interleaving

[0039] Figure 4A shows one embodiment of a process for forwarding read return data. This embodiment is hereinafter referred to as *critical chunk interleaving*. Figure 4B illustrates an example of data transfer according to one embodiment of critical chunk interleaving. In one embodiment, critical chunk interleaving involves interleaving the critical chunks of the cache lines of two read returns, sending the critical chunks in two separate packets, and sending the rest of each cache line in a separate packet. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. Processing logic buffers a chunk of data of a read return from a storage device (processing block 405). Then processing logic checks whether the buffer has any critical chunk not yet forwarded (processing block 410). If the buffer has no critical chunk, then processing logic checks whether another chunk of a cache line is being transferred (processing block 420). If not, then processing logic checks whether enough chunks of data for the read at the top of the read return

queue have been accumulated (processing block 422). If there are insufficient chunks accumulated, processing logic continues to wait for more chunks by repeating processing block 405 (processing block 422). If there are sufficient chunks accumulated, then processing logic starts sending the chunks of the cache line of the read return on the top of the read return queue and indicates that processing logic is transferring a cache line (processing block 424). Processing logic then repeats processing block 405. In one embodiment, processing logic delivers the last chunk in the cache line for the read at the top of the read return queue after the last chunk is ready. For example, referring to Figure 4B, the last chunk of the exemplary cache line 610 is ready at 608.

[0040] On the other hand, if the buffer has no unsent critical chunk and processing logic is transferring a cache line, then processing logic continues with the transfer (processing block 426). Processing logic checks whether all the chunks of the cache line for the read have been transferred (processing block 434). If not, processing logic repeats processing block 405 to wait for the rest of the chunks. Otherwise, processing logic removes the read from the queue and indicates that processing logic is not transferring any cache line (processing block 436). Processing logic then repeats processing block 405.

[0041] If the buffer has an unsent critical chunk, then processing logic checks whether processing logic is transferring a cache line (processing block 430). If so, then processing logic continues with the transfer (processing block 432). Processing logic then checks whether all chunks of the cache line have been sent (processing block 434). If all chunks have been sent, then processing logic removes the read from the queue and

indicates that processing logic is not transferring any cache line (processing block 436). Processing logic then repeats processing block 405.

[0042] If the buffer has a critical chunk not sent yet and processing logic is not transferring any cache line, then processing logic checks whether a header has been sent (processing block 440). If the header has been sent, processing logic gets the critical chunk of the read return on the top of the read return queue and sends the critical chunk on an interconnect (processing block 443). In one embodiment, the interconnect is a packetized interconnect. However, if the header has not been sent, processing logic sends the header and sets the flag "header sent" to 1 (processing block 445). Then processing logic repeats processing block 405.

[0043] Figure 4B shows an example of two cache lines 610 and 620 returned in an overlapping manner from two storage devices in response to two read requests. An example of data transfer according to one embodiment of critical chunk interleaving is shown as 640 in Figure 4B. A header is added to each cache line. For example, the header 646 is added to the cache line from memory channel 0 and the header 648 is added to the cache line from memory channel 1. The critical chunks 642 and 644 of the cache lines 610 and 620 respectively are interleaved. In one embodiment, the critical chunks of two different cache lines are sent in separate packets when they arrive and the remaining chunks of each cache line are sent in two other separate packets. The headers 646 and 648 contain the link level information of the packets transferring the critical chunks 642 and 644 respectively. In one embodiment, the time gap between sending the flits 644 and the non-critical chunks is used to send the flits of a prefetched cache line of

another read data return (not shown) in order to increase the overall efficiency and performance of the system.

[0044] Furthermore, two packet types may be defined to transfer read return data. In one embodiment, the packet types include a critical chunk packet and a cache line packet. Interleaving the critical chunks of separate read returns reduces the latency to the critical chunks of both reads, and hence, improves the performance of many applications. The latency reduction by critical chunk interleaving can be significant when the cache lines returned from the storage devices have not yet queued up in the MCH 220.

Flit-level Interleaving

[0045] Figure 5A shows one embodiment of a process for forwarding read return data. This embodiment is hereinafter referred to as *flit-level interleaving*. In one embodiment, chunks of separate read returns are interleaved and sent as flits on an interconnect. Figures 5B and 5C illustrate examples of data transfer according to various embodiments of flit-level interleaving. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. Processing logic receives a returning read chunk from a storage device in response to a read (processing block 505). Then processing logic checks whether the data chunk belongs to the two read returns on the top of a read return queue (processing block 510). If not, then processing logic buffers the returning chunk (processing block 505). Otherwise, processing logic initializes A to be the read return on the top of the read return queue and B to be the next read return on the top of the read return queue (processing block 520).

[0046] In one embodiment, processing logic assigns Stream to be A if the current flit clock cycle is even (processing block 532). Processing logic assigns Stream to be B, i.e., the second oldest read return, if the current flit clock cycle is odd (processing block 534). Processing logic then checks whether the header of Stream has been sent yet (processing block 536). If not, processing logic sends the header of Stream (processing block 540) and repeats processing block 505. In one embodiment, the header contains link level information of the packet.

[0047] If the header of Stream has already been sent, then processing logic sends the next chunk in Stream (processing block 550). Processing logic then checks whether all chunks in Stream have been sent (processing block 552). If not, processing logic repeats processing block 505. Otherwise, processing logic removes Stream from the read return queue before repeating processing block 505 (processing block 554).

[0048] Figure 5B shows an example of an interleaved stream of chunks of two cache lines generated by flit-level interleaving 630. Examples of data transfer according to one embodiment of critical chunk interleaving, one embodiment of critical chunk with bubble, and store-and-forward are illustrated as 640, 650, and 660, respectively, in Figure 5B. Two cache lines 610 and 620 arrive at the same time from two distinct memory channels in response to two read requests. The flits 632 and 634 containing the critical chunks of the cache lines 610 and 620 respectively are interleaved. Furthermore, two headers 631 and 633 are added, one for each cache line. In addition, the flits 636 and 638 containing the remaining chunks of the two cache lines 610 and 620, respectively, are interleaved to be sent to a processor. In one embodiment, the interleaved flits are sent via an interconnect, which may be a packetized interconnect. It should be apparent to one of

ordinary skill in the art that the flits can be sent to the processor via other means. The latency to both cache lines is reduced because the critical chunks and the remaining chunks are forwarded with less delay.

[0049] Figure 5C shows another example of an interleaved stream 635 of flits of two exemplary cache lines 610 and 625 generated by flit-level interleaving. Unlike the cache lines 610 and 620 in Figure 5B, the cache lines 610 and 625 in Figure 5C do not arrive at the same time. The cache line 625 arrives later than the cache line 610 and partially overlaps with the cache line 610. The header 639 and the chunks of the cache line 610 in Figure 5C are still sent at about the same time as that in Figure 5B. However, there are bubbles of time gaps between the flits containing the header 639 and the first two chunks 632 of the cache line 610 in Figure 5C because the cache line 625 arrives later than the cache line 610. When the cache line 625 starts to arrive at about the same time as the third chunk of the cache line 610, flits containing the header 637 and the chunks 638 of the cache line 625 are interleaved with the flits 636 containing the rest of the chunks of the cache line 610.

[0050] Figure 6A shows the logical representation of one embodiment of a memory controller hub performing flit-level interleaving. Chunks of data are returned from two storage devices, such as the DRAM channels 230 and 240 in Figure 2, in response to two separate reads. The chunks are temporarily stored in the memory channel 0 read return buffer 712 and memory channel 1 read return buffer 714 respectively. The circuitry 730 selects a chunk from the buffers 712 and 714 and forwards the selected chunk to a processor (not shown) via a packetized point-to-point

interconnect 740. In one embodiment, the circuitry 730 includes a slotter, a multiplexer, and a packetizer.

[0051] In one embodiment, each read return is sent in a single packet. The chunks for two read returns sent in two separate packets appear time multiplexed on the interconnect 740. For example, referring to Figure 7, chunks from memory channel 0 are statically assigned to time slot 0 (710) and chunks from memory channel 1 are statically assigned to time slot 1 (720). In one embodiment, a read chunk from a memory channel is dynamically assigned to the first time slot that is open when the chunk becomes available to be forwarded to the interconnect 740. In one embodiment, the assignment remains valid for the transmission of the entire cache line returned in response to the corresponding read. In one embodiment, the idle/busy state of time slots can be maintained in a few bits, which may be updated when new assignments are made and a read transmission completes. Furthermore, it should be appreciated that the flit size may not be equal to the chunk size. If the flit size is larger than the chunk size, the memory controller hub may wait for more data chunk(s) from the memory channels before forming a flit. Alternatively, if the flit size is smaller than the chunk size, more flits are sent for each data chunk.

[0052] The technique disclosed can be extended to an exemplary DRAM system with three memory channels as shown in Figure 6B and 6C. There may be a three-way overlap between the returning read cache lines 2010-2030 from each of the three memory channels. The exemplary system runs on a memory clock signal 2005. The flit clock frequency may be a multiple of the frequency of the memory clock signal 2005. Each

returning read is assigned a time slot and is sent in the assigned time slot. If there is no data returning in a time slot, the time slot may be left empty.

[0053] In one embodiment, the flit clock frequency is three times the frequency of the memory clock signal 2005. Referring to Figure 6B, the two time slots between the flits 2011 and 2012 are left empty because the cache lines 2020 or 2030 have not arrived yet. Same rule applies to the header flit number 2009 and the first data flit 2011 of the first data return. In contrast, one of the two time slots between the flits 2012 and 2013 is assigned to the header 2029 of the cache line 2020 as the first chunk of the cache line 2020 is arriving. The other time slot between the flits 2012 and 2013 is left empty because the cache line 2030 has not arrived yet. The two time slots between the flits 2014 and 2015 are assigned to the header 2039 of the cache line 2030 and the flit 2022, which contains the second chunk of the cache line 2020.

[0054] In one embodiment, the flit clock frequency is twice the frequency of the memory clock signal 2005. Referring to Figure 6C, during the two time slots between the flits 2011 and 2012, which contain the first and second chunks of the cache line 2010, respectively, the header 2029 of the cache line 2020 is sent in the first time slot and the second time slot is left empty because the cache line 2030 has not returned yet. However, during the two time slots between the flits 2012 and 2013, which contain the second and third chunks of the cache line 2010, respectively, the flit 2021 containing the first chunk of the cache line 2020 and the header 2039 of the cache line 2030 are sent in turn. Likewise, during the time slots between the flits 2013 and 2014, which contain the third and fourth chunks of the cache line 2010, respectively, the flits 2022 and 2031 containing

the second chunk of the cache line 2020 and the first chunk of the cache line 2030, respectively, are sent in turn.

[0055] Referring to Figure 6C, the header 2019 of the first cache line 2010 may be sent before the first cache line 2010 starts to arrive, as opposed to the header 2009 in Figure 6B. Likewise, the header 2029 of the second cache line 2020 may also be sent before the second cache line 2020 starts to arrive. The headers (e.g., headers 2019, 2029, etc.) may be sent before the data chunks of the corresponding cache lines arrive because the memory controller can identify when the first data chunk will arrive so as to send the header beforehand.

[0056] An alternate embodiment of a flit-level interleaving in a three-memory channel system is shown in Figure 6D. The interleaving of flits is performed dynamically instead of statically as shown in Figures 6B and 6C. In static interleaving, the flits are interleaved between fixed time intervals. For instance, referring to Figure 6C, a time gap exists between the sixth flit 2036 of the cache line 2030 and the eighth flit 2028 of the cache line 2020 because the eighth flit 2028 of the cache line 2020 is sent at a fixed time after sending the seventh flit of the cache line 2020. In contrast, referring to Figure 6D, the flit 2028 is sent between the flits 2036 and 2037 in order to take advantage of the time gap that would otherwise be left empty as the flits containing chunks of the cache line 2010 have all been sent already. Likewise, the flit 2038 is sent in the time slot right after the time slot assigned to the flit 2037. Dynamic interleaving requires tagging the header and data flits so that the receiver may distinguish which occupies a flit. As illustrated by the example in Figure 6D, dynamic interleaving can provide more efficient

data transfer than static interleaving. However, the implementation of static interleaving may be simpler than dynamic interleaving.

[0057] In general, some embodiments of flit-interleaving are based on a fixed time slot reservation algorithm which can be applied to a system with arbitrary number of memory channels. For a system with *n* memory channels, the interconnect is divided into time slots equal to the period of time to send a flit and time slots are assigned in a round robin fashion amongst all n channels. The time slots are assigned based on the order in which the n channels have data ready to send after the interconnect has been idle. The first channel to have data ready to send after the interconnect has been idle is assigned the next available time slot, say slot i, and every nth timeslot after that, i.e. every slot i, i+n, i+2n, ... until the interconnect is idle once again. Once the interconnect is non-idle, the second channel to have data ready to send is assigned the next available slot that is not already assigned. Supposing that this is slot j, then the second channel is assigned time slots j, j+n, j+2n, ... where j!=i. Similarly, once the interconnect is non-idle, the rth channel to have data ready to send is assigned the next available slot that is not already assigned to the 1^{st}, 2^{nd}, ..., r-1 channels to be assigned time slots. Supposing that this is slot k, then the rth channel is assigned time slots k, k+n, k+2n, ... where k!=j, k!=i, etc For fixed interleaving these time slots assignments remain in effect until no channel has any data to send, at which time the interconnect becomes idle. Once the interconnect becomes non-idle again, the time slots may be reassigned by the same procedure. For dynamic interleaving, such as shown in Figure 6D, the rotation of time slot ownership amongst channels is modulo the number of channels that have data ready to send, rather than modulo n. Whenever a channel changes from not ready to ready to send data or from

ready to not ready to send data, the time slot ownership from that point on is changed to accommodate either one more or one less, respectively, channel in the round-robin ownership. The receiver can detect when such changes occur based on bits that distinguish header flits from data flits, the number of flits in a packet, and the channel assignment contained in the header.

[0058] Furthermore, the technique disclosed can be readily extended to an exemplary DRAM system with four memory channels. In one embodiment, the time axis is divided into the same number of time slots as the number of memory channels in the system. For instance, the time axis may be divided into four time slots when there are four memory channels in the system. However, the time axis in some embodiments may not be divided into the same number of time slots as the number of memory channels. One should appreciate that the technique disclosed is not limited to any particular number of memory channels available in an interleaved memory system. The concept can be applied to systems with a larger number of channels by increasing the speed of the interconnect relative to the memory channel speed. In general, it is easier to increase the interconnect speed than the memory channel speed.

[0059] Furthermore, in one embodiment, the transfer of a read packet header is started after receiving the first chunk for the corresponding read from a storage device. Alternatively, the storage device sends an indication to the MCH earlier so that the MCH can send a header for that read one flit clock cycle before the critical chunk is sent on the interconnect. This approach saves a flit latency for the read return as shown by comparing the cache line 630 with the cache line 660 in Figure 5B.

[0060] The foregoing discussion merely describes some exemplary embodiments of the present invention. One skilled in the art will readily recognize from such discussion, the accompanying drawings and the claims that various modifications can be made without departing from the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting.